EXHIBIT 2

IN THE UNITED STATES DISTRICT COURT
EASTERN DISTRICT OF VIRGINIA
ALEXANDRIA DIVISION

BMG RIGHTS MANAGEMENT (US) LLC, and ROUND HILL MUSIC LP, Plaintiffs, v. COX ENTERPRISES, INC., COX COMMUNICATIONS, INC., and COXCOM, LLC, Defendants.	)	Case No. 1:14-cv-1611 (LO/JFA)

REBUTTAL REPORT OF CHRISTOPHER RUCINSKI

TABLE OF CONTENTS

I. INTRODUCTION AND SHORT SUMMARY OF OPINIONS

1. My name is Christopher Rucinski. I am a computer scientist working at Elysium Digital, L.L.C. ("Elysium Digital"), a technical litigation consulting company located in Boston, MA. I have been retained by counsel for Cox Enterprises, Inc.; Cox Communications, Inc.; and Coxcom, LLC (collectively, "Cox") to respond to the Report of Barbara Frederiksen-Cross ("Frederiksen-Cross"), submitted on June 19, 2015, and to analyze the operation of the source code of Rightscorp, Inc. ("Rightscorp") that has been produced for this litigation.

2. My opinions are described in detail in the subsequent sections of this report, and my opinions can be summarized as follows:

i. Rightscorp's code production is inadequate.

ii. Frederiksen-Cross materially misstates and mischaracterizes the operation of Rightscorp's source code in various ways.

3. This report summarizes my current opinions, which are subject to change depending on ongoing discovery and additional information. A list of the materials I considered in preparing this report is attached as Exhibit B.

II. QUALIFICATIONS AND COMPENSATION

4. I graduated *cum laude* with an A.B. in Computer Science from Princeton University in 2010, and in 2015 I obtained GCFE (GIAC[1] Certified Forensic Examiner) certification. In 2010 I began working at Elysium Digital, and in that capacity I have worked on more than 50 technical matters to date. Many of those technical matters have involved analysis of Java source code, and I have more than ten years of experience writing and analyzing Java source code. I am also well-versed in a number of other programming languages including but not limited to Python, C, Objective-C, Perl, and SQL.

5. I have provided consulting services for the Federal Trade Commission, and I provided deposition testimony in 2013 in the matter of *Shurtape Technologies, LLC et al. v. 3M Company*, U.S. District Court of Western North Carolina (Case No. 5:11-cv-00017) (trademark dispute related to consumer statements on the Internet).

6. My CV is attached as Exhibit A.

7. Elysium Digital is being compensated for my time at my standard rate of $340 per hour. Elysium Digital is also being compensated at varying rates ranging from $120 to $520 per hour for the work of additional Elysium Digital

[1] "GIAC" stands for Global Information Assurance Certification: http://www.giac.org .

employees who are working at my direction. All compensation described above does not depend on the outcome of this case.

III. OPERATION OF RIGHTSCORP'S SOFTWARE

8. This section provides a description of the operation of Rightscorp's software. Rather than reintroduce BitTorrent terminology necessary for this section, I incorporate by reference the section entitled "Salient Aspects of BitTorrent" from William Rosenblatt's July 10th Rebuttal Report ("Rosenblatt"). I agree with the technology discussion in that section and use the same conventions of vocabulary in this report.

9. To provide further clarity on the distinction between "pieces" and "files" and for the sake of brevity, I also incorporate by reference Rosenblatt's Figure 1, which depicts a typical arrangement of a torrent payload where files in the payload are comprised of multiple pieces. Pieces can be thought of as the quantum of exchange between computers from the perspective of the BitTorrent protocol; when peers exchange chunks of a torrent payload, they do so in piece-sized chunks. Files can be thought of as generally comprising one or more pieces; a peer will need to obtain all of the pieces corresponding to a file in the BitTorrent payload before the user can use the file in its entirety.

10. I have also prepared and attached Exhibit C to this report, which provides a high level overview of the operation of Rightscorp's software from the

perspective of the interaction among the various relevant programs and database tables. Exhibit C necessarily omits and simplifies certain details in order to serve as a useful reference.

A. Overview of Rightscorp Software

11. Rightscorp's software processes for detecting instances of alleged copyright infringement and generating notifications of claimed copyright infringement work as follows:

i. Rightscorp claims to receive information about copyrighted music compositions from respective copyright owners. I will refer to this process as "ingestion."

ii. Rightscorp finds and downloads .torrent files from the Internet and matches the metadata appearing in the "info" portion of the .torrent files against the artist and title information for ingested works in order to determine a preliminary list of .torrent files to target.

iii. Rightscorp downloads the torrent payload associated with each such .torrent file and uses some combination of Audible Magic and AcoustID fingerprint technology (and in some cases, manual processes[2]) in an attempt to verify whether information in the names of the song files appearing in the torrent payload appear to

[2] Deposition of Robert Steele, June 11, 2015, 136:15–22.

be accurate. Those .torrent files whose song files appear to be accurately named based on the results of the fingerprint technology are then flagged for further monitoring.

iv. Rightscorp repeatedly connects to the swarm for each such .torrent file and monitors the bitfield returned by IP addresses of the computers in the swarm. If the bitfield indicates that a given IP address is willing to offer at least 10% of the torrent payload to Rightscorp, then Rightscorp records an alleged copyright infringement associated with that given IP address and port number combination for that given .torrent file.

v. Rightscorp multiplies each alleged copyright infringement associated with a given .torrent file by the number of individual songs purportedly in the payload of that .torrent file that Rightscorp is monitoring, creating a multitude of alleged copyright infringements now each associated with an individual song from a torrent payload, rather than just one alleged copyright infringement associated with one .torrent file.

vi. Rightscorp sends one or more automated emails to the Internet Service Provider ("ISP"), e.g. Cox, informing the ISP about each

alleged copyright infringement associated with each file associated with the individual .torrent file.

12. Rightscorp also provides a web-based interface intended for use by ISPs that displays information gathered by its processes for detecting instances of alleged copyright infringement (described above). I understand that Rightscorp calls this interface the "Dashboard."

13. Separate from these processes, Rightscorp sometimes will download and store portions of relevant torrent payloads downloaded from individual IP addresses. These torrent payload portions and the associated download information do not form a part of or impact the processes described above for generating automated emails reporting alleged copyright infringements to ISPs. In particular I have seen no evidence that indicates that these downloaded torrent payloads have any effect on whether emails alleging copyright infringement are sent to any party.[3]

B. **Ingestion of Copyrighted Works**

14. Rightscorp stores information received from copyright owners in the *tracks* database table. This table includes fields for the artist, title, album, label, and owner-provided identifier (if available[4]) of each work provided to Rightscorp for monitoring. This table also includes a timestamp for each work indicating when

[3] Deposition of Greg Boswell, July 3, 2015, 196:12–17.

[4] Deposition of Greg Boswell, July 3, 2015, 51:2–6.

information for that work was first added to the *tracks* database table. This table does not include any actual copyright registration documents that identify the copyright owner and the copyrighted work (in this case, the musical composition).[5]

15. The source code produced by Rightscorp has two methods of adding data to the *tracks* table.[6] The first method is a Java program named `RIAALoader.java`, which reads track information (one track per line) from a text file called "addme.csv" and inserts the information into the *tracks* database table.[7] The second method uses a web-based JSP form defined in `CLT.jsp`[8] that allows a user at Rightscorp to manually enter an artist and a list of songs.[9] After the user submits the form, the code inserts the entered information into the *tracks* database table.[10]

[5] Rightscorp20150605/sqlRequests20150604.txt:453–474, RIGHT_SC00000096.

[6] Deposition of Greg Boswell, July 3, 2015, 148:8–149:3.

[7] Rightscorp20150627/20150627/RIAALoader.java, RIGHT_SC00000347.

[8] During the deposition of Greg Boswell on July 3, several names of files similar to this one were discussed; I have confirmed that `CLT.jsp` is the name of the file that contains the functionality discussed here. See Deposition of Greg Boswell, July 3, 2015, 146:8–148:9.

[9] Deposition of Greg Boswell, July 3, 2015, 146:8–149:3.

[10] Rightscorp20150627/201506027/CLT.jsp, RIGHT_SC00000348–349.

C. Finding .torrent Files on the Internet

16. In order to find .torrent files to check for songs being monitored, Rightscorp uses `KickassTorrentDatabaseLoader.java` to retrieve a list of .torrent files from the web site http://kickass.to. Depending on the first parameter passed to `KickassTorrentDatabaseLoader.java`, a GZipped[11] file will be downloaded from http://kickass.to/hourlydump.txt.gz (approximately 900 KB in size at the time of this writing), http://kickass.to/dailydump.txt.gz (approximately 650 MB in size at the time of this writing), or a URL specified as the second parameter passed to the program. These GZipped files generally contain digests of information about .torrent files; they are updated periodically and made available for download. The text file extracted from the downloaded GZipped file contains one line for each .torrent file. Rightscorp's code reads each line from the text file and inserts the information about the .torrent file into the *knowntorrents* database table.[12]

17. A second program named `MasterTorrentSeeditAll.java` reads information about .torrent files from the *knowntorrents* database table and uses that information to insert certain data about the .torrent file (e.g. a hash of the .torrent file "info" field and the number of pieces) and the .torrent file itself into the

[11] https://en.wikipedia.org/wiki/Gzip .

[12] 20150609/KickassTorrentDatabaseLoader.java, RIGHT_SC00000123.

MasterTorrents database table and certain other data about each file in the torrent payload (e.g. a hash of the corresponding .torrent file "info" field and the size and suggested name of the given piece) into the *MasterTorrentFiles* database table. `MasterTorrentSeeditAll.java` first selects from the *knowntorrents* database table rows corresponding to .torrent files that are not in the "Applications" or "Games" categories and have not been previously processed by `MasterTorrentSeeditAll.java`. For each selected row, it then constructs a URL based on the .torrent hash for the row and the template URLs in the *DownLoadUrls* database table. Next, it downloads the .torrent file from the constructed URL and parses the .torrent file for information about the .torrent file and the torrent payload, such as the hash of the "info" field of the .torrent file, the number of pieces that comprise the payload, and the name and size of each file in the payload. This information is inserted into the *MasterTorrents* and *MasterTorrentFiles* database tables as described above.[13]

D. Filtering Downloaded .torrent Files Based on Matching Artist and Title Against Filename

18. The program `AutoTorrentTransferCode.java` first selects a number of rows from the *tracks* database table. For each such row in the *tracks*

[13] 20150609/MasterTorrentSeeditAll.java, RIGHT_SC00000124–126.

database table, the program then selects all rows from the *MasterTorrentFiles* database table where the path or filename corresponding to the row from the *tracks* database table contains both the artist and title[14] in the row from the *tracks* database table. For each such matching row from the *MasterTorrentFiles* database table, the program copies information from the *MasterTorrents* database table to the *torents* database table and from the *MasterTorrentFiles* database table to the *TorentFiles* database table. New rows are added to the *torents* and *TorentFiles* database tables during this process only if those rows do not yet exist in those tables.[15]

E. **Fingerprinting**

19. Rightscorp attempts to verify the artist and title of files in a torrent payload by using a set of programs that utilize fingerprinting technology to attempt to infer the artist and title metadata. The `VerifyDaemon.sh` program launches the `VerifyDaemon.java` program, which creates a number of threads indicated by the first parameter passed to the program (in this case 100), each of which launches an instance of the `SampleIt3.java` program. These 100 instances of

[14] Other conditions for matching can be used as well, depending on the parameters passed to the `AutoTorrentTransferCode.java` program. I have not seen any code produced by Rightscorp that indicates what the parameters supplied to `AutoTorrentTransferCode.java` are.

[15] Rightscorp archive Jun 1 2015/Rightscorp20150529/AutoTorrentTransferCode/AutoTorrentTransferCode.java, RIGHT_SC00000073–74.

the `SampleIt3.java` program run in parallel. Each instance of the `SampleIt3.java` program sends an HTTP request to `SampleRequest.jsp`. `SampleRequest.jsp` retrieves the first row of a SQL cross join[16] of the *torents* and *TorentFiles* database tables where the torrent hash matches between the rows of the *torents* and *TorentFiles* database tables and the rows have not yet been processed by the `SampleIt3.java` program.[17] `SampleRequest.jsp` returns that row containing information about a torrent payload file and its corresponding .torrent file to `SampleIt3.java`. `SampleIt3.java` then determines which pieces of the torrent payload correspond to the selected file. `SampleIt3.java` uses the jBittorrentAPI library[18] to download those pieces from whichever peers are returned from a hard-coded list of BitTorrent trackers. When the download is complete, `SampleIt3.java` sends the downloaded file to `SFT.jsp`, which inserts a row in the *MusicSamples* database table where the file is put in the "cww" field. Notably, no information about the peers from which the pieces were

[16] See https://en.wikipedia.org/wiki/Join_(SQL)#Cross_join. Essentially this returns a new table where every row in the first table is matched with every row in the second table. If the first table has 3 rows, and the second table has 4 rows, then a cross join of the two tables would result in a table with 12 rows.

[17] The "sampleGuid" field is initially null; it is later set to a non-null value to mark that the corresponding row has been processed by the `SampleIt3.java` program.

[18] http://sourceforge.net/projects/bitext/; this URL appears in the copyright notices for the jBittorrentAPI files that Rightscorp utilizes in its software, e.g. Rightscorp Source Code/jBittorrentAPI/TorrentProcessor.java, RIGHT_SC00000299.

downloaded is recorded; see in particular lines 437–438 of `SampleIt3.java`, which establish hard-coded values for the IP address and the port that would never be encountered for a remote peer.[19]

20. Rightscorp has produced two programs that generate a fingerprint of a song and then attempt to retrieve the artist and title of the song. The `MusicDownload1.java` program uses the Audible Magic program and service, and the `MusicDownLoadPDCleanUP1.java` program uses the AcoustID (also referred to in the Rightscorp source code production as "AqusticID"[20] and "AcusticId"[21]).

21. The `MusicDownLoadPDCleanUp1.java`[22] program retrieves a number of rows for the *TorentFiles* database table where the song corresponding to the given row has not been previously processed by the current program, which is

[19] Rightscorp archive Jun 1 2015/RightscorpRequest20150528/GregScripts/GregScripts /VerifyDaemon.sh, RIGHT_SC00000181; Rightscorp20150605/VerifyDaemon.java, RIGHT_LC00000098; Rightscorp20150605/SampleRequest.jsp, RIGHT_SC00000085–86; Rightscorp20150605/SampleIt3.java, RIGHT_SC00000075–84; Rightscorp20150605/SFT.jsp, RIGHT_SC00000087–89.

[20] e.g. Rightscorp archive Jun 1 2015/Rightscorp20150529/AqusticID and AM/MusicDownLoad1.java, RIGHT_SC00000065–68.

[21] Rightscorp archive June 1 2015/Rightscorp20150529/AqusticID and AM/MusicDownLoadPDCleanUP1.java at line 67, RIGHT_SC00000069–72.

[22] Rightscorp archive June 1 2015/Rightscorp20150529/AqusticID and AM/MusicDownLoadPDCleanUP1.java, RIGHT_SC00000069–72.

indicated by the "songname" field being null or "do not know" and the "enteredintocopyrights" field being 0. For each such row the program retrieves the corresponding file from the *MusicSamples* database table and writes it to a file on the computer running the `MusicDownLoadPDCleanUp1.java` program. The program then runs the fingerprinting executable `fpcalc.exe` to attempt to retrieve the artist and title for the song corresponding to the given row. The getFingerprint[23] method is called at line 67 of `MusicDownLoadPDCleanUp1.java` and the (potentially multiple) results are parsed at lines 80–92. The getFingerprint method may return multiple pairs of artists and titles that match the given song, and the check at line 89 selects the first such artist/title pair where the artist appears in the filename for the given song. If none of the artists returned appear in the filename for the given song, then the last artist/title pair in the list returned from the getFingerprint method is used. The selected artist and title are used to update the *TorentFiles* and *MusicSamplesIndex* database tables. Finally, information from the *TorentFiles* database table is joined with the *tracks* database table based on artist and title, and is copied into the *torrentcopyrights* database table. The join based on artist and title assures that the

[23] Rightscorp archive Jun 1 2015/Rightscorp20150529/AqusticID and AM/AcusticId.java, RIGHT_SC00000059–61.

artist and title returned from the fingerprinting process match the expected artist and title that is recorded in the *tracks* database table.

22. The `MusicDownload1.java` program retrieves a number of rows from the *TorentFiles* database table where the song corresponding to the given row has not been previously processed by the current program, which is indicated by the "songname" field being null and the "enteredintocopyrights" field being 0. For each such row the program retrieves the corresponding file from the *MusicSamples* database table and writes it to a file on the computer running the `MusicDownload1.java` program. This program then runs the appropriate fingerprinting executable `MediaID_offset.exe` to attempt to retrieve the artist and title for the song corresponding to the given row. The cmdProcess[24] method is called at line 51 of `MusicDownload1.java` and the (potentially multiple) results are parsed with a call to the parseAMResponse method at line 57; the parseAMResponse method is defined at line 164. The cmdProcess method may return XML that includes multiple pairs of artists and titles, and those pairs are iterated over in lines 184–195. Unlike in `MusicDownLoadPDCleanUp1.java`, there is no verification at this point that either of the artist and title returned match any string (e.g. filename) associated with the given song. This code will simply

[24] Rightscorp archive Jun 1 2015/Rightscorp20150529/AqusticID and AM/AMMagic2.java, RIGHT_SC00000063–64.

select the last artist/title pair regardless of whether that artist/title pair matches the given song, and that artist and title are used to update the *TorentFiles* and *MusicSamplesIndex* database tables. Finally, information from the *TorentFiles* database table is joined with the *tracks* database table based on artist and title, and is copied into the *torrentcopyrights* database table. The join based on artist and title assures that the artist and title returned from the fingerprinting process match the expected artist and title that is recorded in the *tracks* database table.

23. I have seen nothing in Rightscorp's source code production that unambiguously establishes whether one or both of `MusicDownload1.java` (which utilizes Audible Magic) or `MusicDownLoadPDCleanUp1.java` (which utilizes AcoustID) are currently used to fingerprint any given song. According to deposition testimony of Rightscorp employees, AcoustID is currently used first, and then if AcoustID does not find a match, Audible Magic is used.[25] Finally, if both AcoustID and Audible Magic do not find a match, a human operator will review the song and attempt to identify it.[26]

F. Monitoring Bitfields of IP Addresses

24. The `InfringeBots.sh` program launches the `Deamon.java` program, which creates a number of threads indicated by the first parameter passed

[25] Deposition of Greg Boswell, July 3, 2015, 187:16–188:5.

[26] Deposition of Robert Steele, June 11, 2015, 136:15–22.

to the program (in this case 1,000), each of which launches an instance of the `Test5.java` program,[27] also referred to as "Infringement Finder."[28] These 1,000 instances of the `Test5.java` program run in parallel. Each instance sends an HTTP request to `TorrentRequest.jsp`, which selects a random row from the *torrentcopyrights* database table and retrieves information about the corresponding .torrent file from the *torents* database table, then returns that information to `Test5.java`.

25. `Test5.java` then uses the jBittorrentAPI library to get a list of peers from the tracker associated with the returned .torrent file and initiate a connection to each peer. As part of the connection initiation, the peer sends a bitfield indicating which pieces of the torrent payload the peer is willing to offer. After thirty minutes of collecting a list of peers from the tracker, `Test5.java` sends information about each peer, including the peer's IP address and port number, the bitfield sent by the peer, the date and time at which the peer was contacted,[29] and a randomly-generated UUID, to `Detected.jsp`, which inserts the

[27] Rightscorp archive Jun 1 2015/RightscorpRequest20150528/GregScripts/GregScripts/InfringeBots.sh, RIGHT_SC00000145; Rightscorp Source Code/com/boswell/torrent/Deamon.java, RIGHT_SC00000020–21.

[28] Deposition of Greg Boswell, July 3, 2015, 199:6–10.

[29] See the constructor of the Peer class – Rightscorp Source Code/jBittorrentAPI/Peer.java, RIGHT_SC00000278.

information into the *TorrentInfractions* database table. In particular, the "fullfile" field in the *TorrentInfractions* database table is set to true if each and every bit in the bitfield sent by the peer was set to 1; otherwise the "fullfile" field is set to false (see `Test5.java`, ll. 212–222; `Detected.jsp`, ll. 46, 79, 87).

26. According to the Rightscorp source code that I received in paper form on July 8, 2015, and in electronic form on July 9, 2015, Rightscorp periodically reevaluates the "fullfile" field for an entry in the *TorrentInfractions* database table after `Test5.java` creates the entry through `Detected.jsp`. Through this process, the *TorrentInfractions* database table is updated such that the "fullfile" field is set to 1 if the bitfield contains 1 for as few as 10% of the pieces of the torrent payload, overwriting the value that had previously been determined by the `Test5.java` program. In calculating the 10% threshold, the source code does not distinguish between pieces corresponding to songs that Rightscorp is monitoring or not monitoring and does not even distinguish between pieces that correspond to

audio files and other types of information (e.g. image files or text files). This update to the *TorrentInfractions* database table is performed every 15 minutes.[30]

27. Each row in the *TorrentInfractions* database table corresponds to a single .torrent file rather than an individual file within a torrent payload. Therefore, regardless of the total number of files in a particular torrent payload that Rightscorp is attempting to monitor, a single entry will be made in the *TorrentInfractions* database table once any one of those files is selected and at least one peer is monitored as part of the execution of the `Test5.java` program and the subsequent reprocessing of an entry based on the 10% bitfield threshold.[31]

28. The selection of the row from the *torrentcopyrights* database table in `TorrentRequest.jsp` is random, and multiple instances of `Test5.java` are executing simultaneously (see ¶24). `TorrentRequest.jsp` does not include any check for the last time that a given row in the *torrentcopyrights* database table was selected for processing by `Test5.java`. Because of this, two instances of

[30] See FullFilesBeing.txt, RIGHT_SC00000402 and FullFileFix.txt. FullFileFix.txt calls a stored procedure named "ReEvaluateFullFilesBeing0", but the stored procedure defined in FullFilesBeing.txt is named "ReEvaluateFullFilesBeing0Hold". In my analysis I have assumed that this is an accidental error in the production and that FullFilesFix.txt governs when the ReEvaluateFullFilesBeing0Hold stored procedure is run, but I reserve the right to supplement my opinions if it is determined that this discrepancy is not a production error.

[31] Rightscorp Source Code/com/boswell/torrent/Test5.java, RIGHT_SC00000035–39; 20150609/TorrentRequest.jsp, RIGHT_SC00000134–135; Rightscorp archive Jun 1 2015/RightscorpRequest20150528/Detected.jsp, RIGHT_SC00000138–139.

`Test5.java` running at different times on the same day could select the very same row (via `TorrentRequest.jsp`), resulting in the same .torrent file being processed by two instances of `Test5.java` on the same day. It could even happen that the first instance of `Test5.java` is still running on that day when the second instance of `Test5.java` starts to process the same .torrent file. In either of these circumstances superfluous rows will be inserted into the *TorrentInfractions* database table corresponding to the superfluous instance of `Test5.java` targeting the same .torrent file on the same day.

G. Expanding Alleged Infringements Associated with .torrent Files to Alleged Infringements Associated with Individual Songs

29. The `TorrentInfractionExpander.java` program selects rows from a join of the *TorrentInfractions* and *torrentcopyrights* database tables. Because of the join, multiple rows are returned for each row in the *TorrentInfractions* database table, with one row for each file listed in the *torrentcopyrights* database table that is purportedly in each torrent payload associated with the row from the *TorrentInfractions* database table. Thus, if a torrent payload associated with a row in the *TorrentInfractions* database table contains 100 songs that are being monitored by Rightscorp, the `TorrentInfractionExpander.java` program will get 100 rows for that

torrent payload. For each row that `TorrentInfractionExpander.java` gets, it then inserts a row into the *ExpandedTI* database table.[32]

30. The `ExpanderToInfractions.java` program selects rows from the *ExpandedTI* database table. For each selected row, the program inserts a row into each of the *EmailRegestry*, *infractions*, and *TemporaryHolding* (used with `SampleIt2.java`, as discussed below) database tables. The *infractions* database table is used to generate notifications of claimed infringement (explained below).[33]

H. Generating Notifications of Claimed Infringement

31. Notifications of claimed infringement are generated by the `CEmail.java` program. This program reads the email template contained in the `cox.txt` file. The program then selects a number of rows from a combination of the *infractions*, *copyrights*, and *EmailRegestry* database tables.[34] For each selected row, the program replaces a number of placeholders such as "%title%", "%artist%", and "%ip%" with the corresponding values from the database tables and sends the email to the listed ISP email address (for Cox, the email address is "abuse@cox.net"). After sending the email, in order to note that an email has been

[32] 20150609/TorrentInfractionExpander.java, RIGHT_SC00000132–133.

[33] 20150609/ExpanderToInfractions.java, RIGHT_SC00000099–101.

[34] The default limit to the number of rows selected at one time is 200, though there does not appear to be any other limitation on which rows are selected, such as excluding more than one row for the same combination of IP address, port number, song, and date.

sent for that infraction, the program updates the *infractions* database table and inserts the email into the *Emailed* database table.

32. Note that because an email is generated for each *file* monitored by Rightscorp in a torrent payload and a torrent payload may contain hundreds of individual songs (e.g., if the payload contains the complete discography for a particular artist), dozens or even hundreds of emails may be generated as a result of a single IP address participating in a single swarm for a brief amount of time. Rightscorp treats the offer of a *single* torrent payload, based on its 10% bitfield thresholding, as an individual copyright infringement of *each* of the songs monitored by Rightscorp purportedly contained within the payload. I have seen no source code that limits the number of emails that are sent by this process for a given song over any timeframe.[35]

33. Termination requests are generated by the `AutoTermRequest.jsp` program. This program reads the email template contained in the `termrequest2.txt` file. This program retrieves information from several database tables and replaces a number of placeholders such as "%Isp%", "%Title%", "%started%", "%finished%", and "%totals%" with the corresponding values from the database tables. This program then selects rows from

[35] 2015.05.26 – Additional Rightscorp Source Code/CoxEmailCode/CEmail.java, RIGHT_SC00000017–19; 2015.05.26 – Additional Rightscorp Source Code/CoxEmailCode/cox.txt, RIGHT_SC00000009–11.

the *InfMultiSummary* table where a combination of IP address and port number has at least five alleged copyright infringements across a span of at least three days.[36] Information from each of these rows, such as IP address, port number, and number of alleged infringements, is added to a comma-separated value file that is attached to an email. Finally, the program sends the email to the ISP email address listed in the *AutoTermContactList* database table, adding "stlrbrt@yahoo.com" and "gboswell45@yahoo.com" as BCC recipients.[37]

I. ISP Dashboard

34. Rightscorp also produced code for a web-based interface that displays information gathered by its processes for detecting instances of alleged copyright infringement and appears intended for use by ISPs. I understand that Rightscorp calls this interface the "Dashboard," so I will use the same terminology.

35. As part of this Dashboard, Rightscorp produced code for populating a web page titled "Multiple Infringers"; I have analyzed the source code for this web page, which is in `multipleinfringers.jsp`.

[36] Greg Boswell testified that this threshold of five infringements over three days used to be "substantially higher," which suggests that more IP addresses are currently categorized as alleged repeat infringers (Deposition of Greg Boswell, July 3, 2015, 125:11–126:6).

[37] 2015.05.26 – Additional Rightscorp Source Code/CoxEmailCode/AutoTermRequest.jsp, RIGHT_SC00000012–16; 2015.05.26 – Additional Rightscorp Source Code/CoxEmailCode/termrequest2.txt, RIGHT_SC00000008.

`multipleinfringers.jsp` uses information from the *InfMultiSummary* database table, which is populated with information from the *infractions* and *torents* database tables. Specifically, the *InfMultiSummary* database table contains one row per combination of IP address and port number of an alleged infringer, with each row including information such as the number of rows in the *infractions* database table for the IP address and port number and the earliest and latest dates of alleged infringement. `multipleinfringers.jsp` displays two sets of data. The first set has three rows which show the number of alleged infringers and the number of alleged infringements. The first row is the totals for alleged infringers who have paid Rightscorp, the second row is for alleged infringers who have not paid Rightscorp, and the last row is the totals for the first two rows. The second set of data on the page lists rows from the *InfMultiSummary* database table, showing the IP address, port number, number of alleged infringements, latest and earliest dates of alleged infringement, the number of days between the earliest and latest dates of alleged infringement, and whether or not the alleged infringer has paid Rightscorp. Neither of the tables shown as part of this Dashboard limit (either by time period or absolute number) the amount of alleged infringements included in the data that is displayed.[38]

[38] Rightscorp Source Code/coxstuff/MultiInfringement.txt, RIGHT_SC00000052; Rightscorp Source Code/coxstuff/multipleinfringers.jsp, RIGHT_SC00000053–55.

J. Downloading and Storing Torrent Payload Files

36. The "tracker" stored procedure is run manually[39], and it copies information from the *TemporaryHolding* database table (discussed above) into the *GetThisInfringer* database table, which is used by the `SampleIt2.java` program.[40] The `Deamon.sh` program launches the `SampleDaemon.java` program, which creates a number of threads indicated by the first parameter passed to the program (in this case 100), each of which launches an instance of the `SampleIt2.java` program. These 100 instances of the `SampleIt2.java` program run in parallel. The `SampleIt2.java` program is executed independently of other programs in Rightscorp's system, such as the fingerprinting programs and the programs that generate notifications of claimed infringement.[41] Each instance of the `SampleIt2.java` program sends an HTTP request to `DownloadRequest.jsp`, which selects a random row from the *GetThisInfringer* database table joined with the *torents* database table and sends the IP address, port number, file name, and .torrent file back to `SampleIt2.java`. `SampleIt2.java` then determines which pieces of the torrent payload correspond to the selected file, connects to the peer at the selected IP address and

[39] Deposition of Greg Boswell, July 3, 2015, 197:8–198:2.

[40] "tracker" stored procedure, 20150609/sqs20150609, RIGHT_SC00000128–129.

[41] Deposition of Greg Boswell, July 3, 2015, 196:12–17.

port number, and downloads the pieces for the file. After the download is complete, `SampleIt2.java` sends the downloaded file to `TFT.jsp`, which inserts the downloaded file into the *TorrentSamples* database table.[42]

37. The `SampleIt2.java` program is not involved with the generation of notifications of claimed infringement in any way.[43] Its operation depends upon the "tracker" stored procedure being executed so that the *GetThisInfringer* database table will be populated. The *GetThisInfringer* database table is populated with information from the *TemporaryHolding* database table, which is populated with information from the *ExpandedTI* database table. The *ExpandedTI* database table is used to populate the *infractions* database table, which itself is used to generate notifications of claimed infringement. The generation of notifications of claimed infringement happens in a separate process from `SampleIt2.java`.

IV. RIGHTSCORP'S SOURCE CODE PRODUCTION IS INADEQUATE

38. Rightscorp's source code production has generally been inadequate. Due to this inadequacy, I have not been able to perform timely analysis on much of

[42] Rightscorp archive Jun 1 2015/RightscorpRequest20150528/GregScripts/GregScripts/Deamon.sh, RIGHT_SC00000144; Rightscorp Source Code/com/boswell/torrent/SampleDaemon.java, RIGHT_SC00000027–28; Rightscorp Source Code/com/boswell/torrent/SampleIt2.java, RIGHT_SC00000029–34; Rightscorp archive Jun 1 2015/RightscorpRequest20150528/DownloadRequest.jsp, RIGHT_SC00000140; 20150609/TFT.jsp, RIGHT_SC00000130–131.

[43] Deposition of Greg Boswell, July 3, 2015, 196:12–17.

the code that was eventually produced, and I have not been able to analyze code that was operational during all relevant times. In this section I enumerate the various ways that Rightscorp has failed to produce all relevant requested source code in a reasonably timely manner and in a reasonable format.

39. As a general matter, Rightscorp has produced only a single version of each source code file that it has produced; Rightscorp has not produced any historical versions of any source code files. During his deposition, Greg Boswell stated that Rightscorp does not keep any logs of changes made to its source code, nor does it keep past copies of its source code.[44] It is a fundamental best practice[45] for professional software developers to maintain historical versions of source code using any one of several freely available revision control systems[46] (e.g. Git or Subversion) in order to facilitate software development in a variety of ways. It is unreasonable for Rightscorp not to have used such a system.

40. Because Rightscorp does not use a revision control system, Rightscorp has been able to provide only current versions of any source code produced during the course of this litigation. Therefore, it is not possible for me to

[44] Deposition of Greg Boswell, July 3, 2015, 102:5–103:9.

[45] See, e.g., "Becoming a Better Programmer: A Handbook for People Who Care About Code" by Peter Goodliffe, page 253: https://books.google.com/books?id=Ab6uBAAAQBAJ&lpg=PT253&pg=PT253 .

[46] https://en.wikipedia.org/wiki/Revision_control .

determine how Rightscorp's software operated with respect to previous versions of Rightscorp's software.

41. Because Rightscorp does not use a revision control system, there is also no way for me to know exactly when the current version of each produced source code file was used and/or implemented. Some produced source code files include a date in a comment[47], but such dates do not reliably indicate the date a file was created or last modified, as comments oftentimes tend to not be updated to reflect changes in source code.[48] Many of the produced source code files do not even include such a date in the comments.[49] In particular, `FullFilesBeing.txt` (discussed below) does not include any date in any of its comments.

42. I understand that Plaintiffs assert infringement of the works at issue as far back as 2012. Because of Rightscorp's failure to implement a revision control system, I am unable to determine whether the operative source code that Rightscorp had in place during the relevant time period was in any way different from the version of source code that Rightscorp has produced.

[47] e.g. Rightscorp Source Code/com/boswell/torrent/Deamon.java, RIGHT_SC00000020.

[48] e.g. Rightscorp Source Code/com/boswell/torrent/SampleIt2.java, RIGHT_SC00000029, line 18: " * Demo code. In Development".

[49] e.g. Rightscorp archive Jun 12015/Rightscorp20150529/20150529/TFT.jsp.

43. `FullFilesBeing.txt` defines the threshold of 10% for the number of bits in a bitfield that are required to be advertised by a BitTorrent peer at an IP address before Rightscorp's software will consider the peer to be infringing. `FullFileFix.txt` specifies that `FullFilesBeing.txt` is executed every fifteen minutes.[50] I received `FullFilesBeing.txt` in paper form on July 8, 2015, and in electronic form on July 9, 2015. I received `FullFileFix.txt` in electronic form on July 9, 2015. I received these two files only after testimony from Greg Boswell led to the conclusion that these relevant files had not been produced.[51] But for that testimony, these files might not have been produced at all, and an analysis of the other produced source code would have erroneously led one to believe that the threshold was instead 100% as indicated by the `Test5.java` program. Neither of these files are mentioned in Frederiksen-Cross, and Frederiksen-Cross does not note the omission of these files from the production at the time Frederiksen-Cross was submitted. Indeed, Frederiksen-Cross states at ¶56,

[50] FullFileFix.txt calls a stored procedure named "ReEvaluateFullFilesBeing0", but the stored procedure defined in FullFilesBeing.txt is named "ReEvaluateFullFilesBeing0Hold". In my analysis I have assumed that this is an accidental error in the production and that FullFilesFix.txt governs when the ReEvaluateFullFilesBeing0Hold stored procedure is run, but I reserve the right to supplement my opinions if it is determined that this discrepancy is not a production error.

[51] Deposition of Greg Boswell, July 3, 2015, 60:11–18; 280:6–281:10.

"By verifying that the bitfield contains a '1' for every piece of the desired file Rightscorp can confirm whether the Peer is a Seed offering the entire file."

44. As discussed above in ¶23, no source code has been produced that indicates which of the fingerprinting services between Audible Magic and AcoustID is used before the other or if both are used.

45. During his deposition, Greg Boswell stated that Rightscorp counts only one infringement per file per day for a given IP address and port number.[52] However, I have not seen anything in Rightscorp's source code production that supports this statement. On the contrary, I have seen an instance where five infringement notices were generated for the same IP address (174.64.13.245), port number (46344), file (205 – Antenna head.mp3), and .torrent file (with a .torrent file info hash of D381142DFF09D891B81EB0EBAFFD9009DE05AD01) during the same day (February 16, 2012); those five infringement notices are RGHTS10576075, RGHTS10578537, RGHTS10579202, RGHTS10580459, and RGHTS10585268. Each of these five infringement notices were generated by monitoring the same swarm associated with a .torrent file as evidenced by the same .torrent file info hash recorded in RGHTS00011291_STEPRIGHT00006233.csv for each of these five infringement notices.

[52] Deposition of Greg Boswell, July 3, 2015, 68:14–72:6.

46. As I describe above in ¶31, the `CEmail.java` program generates notifications of claimed infringement. However, I have not seen any code in Rightscorp's production that invokes this program.

V. FREDERIKSEN-CROSS MISSTATES AND MISCHARACTERIZES THE OPERATION OF RIGHTSCORP'S SOURCE CODE

47. Frederiksen-Cross misstates and mischaracterizes certain aspects of the operation of Rightscorp source code, and in this section I describe those discrepancies. The absence of discussion of any parts of Frederiksen-Cross from this section should not be interpreted to mean that I agree with those parts.

A. SampleIt3.java, SampleIt2.java, and Flow of Information Through Rightscorp's Software

48. Frederiksen-Cross does not discuss the `SampleIt3.java` and `SampleIt2.java` programs in a way that is consistent with their respective operation. Adding to the confusion is that there is no single "SampleIt" program, though Frederiksen-Cross repeatedly refers to it as such. `SampleIt3.java` downloads a file in a torrent payload from arbitrary peers in a swarm so that the file can be fingerprinted. `SampleIt2.java` downloads a file in a torrent payload from an individual peer in a swarm but does not invoke or influence the fingerprinting process. Neither `SampleIt3.java` nor `SampleIt2.java`

directly relate to detecting instances of alleged infringement or generating notifications of claimed infringement. More detail is provided below.

49. Frederiksen-Cross states at ¶53, "This value serves as a flag to tell the Rightscorp SamplelIt process which torrents to sample when the system is preserving downloaded songs as evidence." Frederiksen-Cross also states at ¶67, "Rightscorp uses software that prepares digital fingerprints from the content of the files Rightscorp captures via the SampleIt process." "SampleIt" is ambiguous in both these cases in that it could refer to SampleIt2 or SampleIt3; there is no process in the Rightscorp code production that is simply called "SampleIt".

50. Frederiksen-Cross states at ¶60, "Each file that Samplelt3 collects is downloaded in its entirety from a single Peer, and Samplelt3 saves the entire file that it downloads in a database along with an evidentiary record documenting the file download." This is incorrect because `SampleIt3.java` does not download files from a specific peer. For example, referring to "Rightscorp20150605/SampleIt3.java" (RIGHTS_SC00000079), line 314 contains the statement "dm.startTrackerUpdate();". "dm" is an instance of TrackManager (see line 218), which extends the DownloadManager class (see line 516). "startTrackerUpdate()" is implemented in "Rightscorp Source Code/jBittorrentAPI/DownloadManager.java" (RIGHT_SC00000225) at line 200 and utilizes multiple peers to download a torrent payload. Contrast the statement of

"dm.startTrackerUpdate();" with the statement of "dm.connect(new Peer("foo", ip, Integer.parseInt(port)));" at line 175 of "Rightscorp Source Code/com/boswell/torrent/SampleIt2.java" (RIGHT_SC00000029). The latter statement indicates setting up a connection with an individual peer while the former indicates setting up a connection with arbitrary peers in the swarm. In contrast, `SampleIt2.java` does download a file from a specific peer.

51. Frederiksen-Cross states at ¶60 during the discussion of SampleIt3, "The record created by this process includes the a unique identifying ID for the record, the torrent file hash, Torrent file name, IP address and port, date and time, bitmap field showing the offer to make available the file pieces, and abuser's ISP." As stated in the preceding paragraph, `SampleIt3.java` does not download files from a specific peer, so there is no corresponding specific IP address, port, bitmap field, or ISP to record. Lines 434 through 483 of "Rightscorp20150605/SampleIt3.java" (RIGHT_SC00000075) create an XML string and send that string to `SFT.jsp` to be recorded in the *MusicSamples* database table. That XML string does contain an IP address and a port, which are set to hard-coded values of "127.0.0.1" (localhost) and "0" respectively. Both of these values would never be encountered during an actual download from a peer in a swarm. That XML string does not contain a bitmap field or ISP.

52. Frederiksen-Cross states at ¶62, "For efficiency reasons, Samplelt3 does not collect a copy of every file identified by Infringement Finder, but instead downloads samples from a predetermined ratio (e.g. 1:5000) of Peers on each run." This sentence is incorrect in two ways.

i. First, `SampleIt3.java` retrieves a file to download from a cross join of the *torents* and *TorentFiles* database tables via `SampleRequest.java` (see ¶19). Infringement Finder (`Test5.java`) does not populate either of those tables; rather it populates the *TorrentInfractions* database table (see ¶25 and Frederiksen-Cross at ¶57). Before `SampleIt3.java` operates on a file, that file has not been, and may not ever be (depending on the output of `SampleIt3.java`), processed by Infringement Finder (`Test5.java`).

ii. Second, there is no reference in `SampleIt3.java` (or `SampleIt2.java`) to a ratio of 1:5000 or any other ratio that affects the number of files that are downloaded. Frederiksen-Cross provides no citation for this ratio, and it is therefore unclear what evidence, if any, supports its existence. Greg Boswell has also

testified that there is no ratio in any Rightscorp code that affects the number of files that are downloaded by this program.[53]

53. Frederiksen-Cross states at ¶62, "SampleIt3 also limits the samples it collects to a maximum of two songs per Peer per SampleIt3 run." I have found no evidence in `SampleIt3.java` that a maximum of two songs are collected per run. The closest thing to this that I found is the code at line 181 of "Rightscorp20150605/SampleIt3.java" (RIGHTS_SC00000075), which states 'System.out.println("looking for 2 " + filename);', but that statement just precedes two code blocks that attempt to match a filename two different ways and have nothing to do with limiting the number of songs collected.

54. Having now clarified the operation of `SampleIt3.java`, an important point is that Frederiksen-Cross makes it easy to misunderstand the overall workflow of the Rightscorp software by arranging descriptions of the various programs in an order that does not reflect how information is processed by the system. In Section III of this report I have described the various programs in the order in which information flows from one component to the next. Frederiksen-Cross describes the operation of Infringement Finder (`Test5.java`) in ¶¶56–59, then describes the operation of the Music Sampling Components (`SampleIt2.java` and `SampleIt3.java`) in ¶¶60–64, then describes the

[53] Deposition of Greg Boswell, July 3, 2015, 249:11–250:4.

operation of the File Verification Components (including Audible Magic and AcoustID) in ¶¶65–71, and then finally the Notice Generating Components in ¶¶72–79. As I describe in Section III of this report, information generated by the ingestion process is used by `SampleIt3.java`, information generated by `SampleIt3.java` is used by programs employing Audible Magic and AcoustID fingerprinting services, information generated by the programs employing Audible Magic and AcoustID is used by `Test5.java`, and information generated by `Test5.java` is used by the programs used to generate notifications of claimed infringement and separately by `SampleIt2.java`.

55. Rightscorp's software does not download all or portions of files from individual peers before sending out notifications of claimed infringement to any party. As discussed earlier in this report, the download of an entire file from an individual peer is done by the `SampleIt2.java` program. The `SampleIt2.java` program utilizes information generated by the `Test5.java` program and therefore `Test5.java` must operate on a given .torrent file *before* `SampleIt2.java` can subsequently operate to download a file in the torrent payload corresponding to that .torrent file. The `SampleIt2.java` program is not

involved with the generation of notifications of claimed infringement in any way.[54] A detailed explanation is provided in ¶¶36–37 above.

56. Frederiksen-Cross states at ¶66, "Rightscorp submits queries to the AcoustlD database to identify the specific tracks (artist and title) collected via the sampling processes (Samplelt2 and Samplelt3)." This statement is incorrect because `SampleIt2.java` is not involved in the fingerprinting process.[55] Rather, as discussed earlier in this report, `SampleIt2.java` is used to download individual files from specific IP addresses, independent from the process of generating notifications of claimed infringement. Greg Boswell has testified that the `SampleIt2.java` program is manually invoked and that he could recall it being invoked on only two occasions, once to target Cox customers and once to target "Grande ISP" customers.[56]

57. Frederiksen-Cross states in ¶¶54–55 that the programs `TorrentInfractionExpander.java` and `ExpanderToInfractions.java` operate on .torrent files before the Infringement Finder (`Test5.java`) program operates on them. However, as stated earlier in this report at ¶¶29–30 above, the programs

[54] Deposition of Greg Boswell, July 3, 2015, 196:12–17.

[55] Deposition of Greg Boswell, July 3, 2015, 196:12–17.

[56] Deposition of Greg Boswell, July 3, 2015, 251:4–252:10.

`TorrentInfractionExpander.java` and `ExpanderToInfractions.java` actually operate on information that is output by `Test5.java` in order to prepare to send email notifications of claimed infringement for each song monitored by Rightscorp purportedly contained in a torrent payload.

58. Frederiksen-Cross states at ¶70, "Together these products allow Rightscorp to identify the files downloaded via SampleIt2 and SampleIt3 and to confirm whether the songs Rightscorp downloads are copies of Rightscorp customers' protected works." As described in ¶¶19–23 above, Rightscorp's software uses the Audible Magic and AcoustID fingerprinting software to attempt to identify the artist and title of a file downloaded by the `SampleIt3.java` program. However, at no time are the specific files downloaded by the `SampleIt2.java` program (which downloads files from individual IP addresses) submitted to any fingerprinting system, nor is `SampleIt2.java` involved in the generation of notifications of claimed infringement.

B. **Fingerprinting**

59. When discussing fingerprinting Frederiksen-Cross glosses over the possibility that each fingerprinting service might return multiple artist and title matches for any given song.

60. Frederiksen-Cross states at ¶68, while discussing `MusicDownload1.java`, "Upon return, the testSampleRequest function to parse the XML file to get the data fields returned, which include the artist and title of the song." Frederiksen-Cross does not state how this artist and title value are determined from a potential list of artist/title pairs. I discuss this and the `MusicDownload1.java` program in detail in ¶22.

61. Frederiksen-Cross states at ¶69, while discussing `MusicDownLoadPDCleanUp1.java`, "The testSample[Request][57] function then calls the function 'requestAcoustlD' to send a request to the web server 'api.acoustid.org' passing the file's fingerprint and receiving back an XML file that includes the artist and title." This statement is misleading insofar as it suggests that the received XML file is likely to include a single artist and title. Referring to `MusicDownLoadPDCleanUp1.java`, line 71, in the "testSampleRequest" function, calls the "requestAcoustID" function. Lines 80 through 92 iterate through the returned pairs of artist and title; this iteration is necessary because the "requestAcoustID" function returns multiple possible artist and title matches for the song file that it is given. I also discuss the operation of this program in ¶21.

[57] As a matter of clarity, there is no "testSampleId" function in `MusicDownLoadPDCleanUp1.java`, though there is a "testSampleRequest" function, and I will assume based on the preceding sentences in Frederiksen-Cross at ¶69 that that is the function that was intended to be named.

C. Bitfield Threshold for Detecting Alleged Infringements

62. Frederiksen-Cross states at ¶56, "By verifying that the bitfield contains a '1' for every piece of the desired file Rightscorp can confirm whether the Peer is a Seed offering the entire file. After Infringement Finder receives the bitfield it creates an evidentiary Infringement Record." According to the Rightscorp source code that I received in paper form on July 8, 2015, and in electronic form on July 9, 2015, notifications of claimed infringement are generated even if the bitfield contains a 1 for as little as 10% of the pieces in the torrent payload.[58] Frederiksen-Cross does not mention this file or the procedure contained in this file, nor is there any indication that Ms. Frederiksen-Cross saw or was aware of this source code. Greg Boswell also testified that the current operation of Rightscorp source code requires only 10% of the bits in the bitfield to be set to 1 in order for Rightscorp software to list a given IP address as infringing.[59] Because Rightscorp currently targets peers offering to disseminate a minimum of 10% of a torrent payload, it is not certain nor is it necessarily even likely that these peers are Seeds, as defined in Frederiksen-Cross at ¶37: "**Seed** – A Peer who has the entire file and is offering the file content for upload to other Peers on the network" (emphasis in original).

[58] FullFilesBeing.txt, RIGHT_SC00000402, which is a paper copy produced by Rightscorp. An electronic copy was provided on July 9, 2015.

[59] Deposition of Greg Boswell, July 3, 2015, 60:11–25.

D. Other Misstatements and Mischaracterizations

63. Frederiksen-Cross mischaracterizes the operation of the Rightscorp source code in a few other ways that might lead to confusion.

64. Frederiksen-Cross states at ¶49, "One table (MasterTorrents) contains a copy of the entire [].torrent file and the other table (MasterTorrentFiles) stores each piece entry in the [].torrent file as a record." The *MasterTorrentFiles* database table does not store the actual contents of the pieces of torrent payloads nor does it store metadata about the pieces of torrent payloads. Rather, it stores metadata about the files contained in torrent payloads that appears in the "info" portion of a .torrent file. For example, referring to "20150609/MasterTorrentSeedItAll.java" (RIGHT_SC00000124–126), line 153 creates a SQL statement to insert data into the *MasterTorrentFiles* database table. Line 160 sets the "filename" column to the value "t.name.get(PiecePointer).toString()". Despite the variable name "PiecePointer", this in fact refers to the filename of the current file in the torrent payload. Lines 64 through 67 of "Rightscorp Source Code/jBittorrentAPI/TorrentFile.java" establish that the "name" property of the TorrentFile "contains the path of the file to be saved", so "t.name.get(PiecePointer)" discussed above returns the path for the file at the index of "PiecePointer".

65. Frederiksen-Cross states at ¶53, "After [].torrent file [sic] have been collected and stored in the Rightscorp database, a program called `AutoTorrentTransferCode.java` builds a query for the "tracks" table to obtain a list of protected works." This characterization glosses over one important detail about the overall operation of the Rightscorp software. The query used here matches only artist and title against the path and filename of files in the torrent payloads. This is discussed in detail in ¶18.

VI. FURTHER WORK

66. I may prepare visual aids to demonstrate various aspects of my testimony at trial. I may also review additional materials produced or otherwise prepared by parties or experts in this case. I reserve the right to supplement my opinions if Plaintiffs or their experts or other parties related to this case produce new materials or source code and/or if Court decisions require me to clarify or provide additional support for my opinions. I also reserve the right to supplement this report based on additional documents or information, including but not limited to additional deposition testimony, additional claims asserted by Plaintiffs, or if Plaintiffs' experts offer any opinions.

Date: July 10, 2015

Christopher Rucinski

EXHIBITS A-C

INTENTIONALLY OMITTED